REACHING OUT:
off-the-shelf embedded-network connectivity

IS IT REALLY AS EASY TO NETWORK-ENABLE EMBEDDED DEVICES AS MANY VENDORS CLAIM?

Illustration by Federico Jorden

FOR YEARS, you've heard fantastic stories about network-enabled refrigerators and toasters—stories that, from my perspective, have only reduced the credibility of embedding connectivity. However, as the cost of connectivity continues to drop, an increasing number of embedded applications can make serious enough use of such technology to justify the extra expense (**Reference 1**).

For this hands-on project, I wanted to see for myself just how ready embedded TCP/IP (Transmission Control Protocol/Internet Protocol) is for mass deployment. Given that myriad companies offer networking chips and TCP/IP stacks, I chose to work with several kits to give myself multiple data points. I chose kits from Connect One, Cyan (using a stack from CMX), Rabbit, Ubicom, and Zilog. My goal was to determine whether embedded TCP/IP is truly affordable, in both dollars and development time, what I could expect from vendors, and which processor/software combination promises to best meet the constraints of an application.

Quickly getting the evaluation boards up and running was critical. Foremost, I didn't want to spend a lot of time learning a system just to evaluate it. Some of the technical manuals that come with these kits are daunting; you shouldn't have to read 300 pages just to blink an LED or serve up a "Hello World!" page. I also wanted to be able to evaluate several kits to increase the number of processors from which I could make my final selection. I wanted to take a tour of each platform, quickly loading a demo application, making changes that would better reflect the conditions of my application, verifying performance, and getting an idea of the usefulness of the available tools.

As I worked with the kits, I realized that the price of the processor and software, although important, probably wasn't going to be the most important consideration in my final selection. In general, the vendors seemed fairly competitive on price—in the ballpark of $10 for the processor in volume quantities, including the cost of the stack and development tools. Next, I looked at performance and code footprint, which the vendors often tout as the primary considerations for selecting a processor and stack. However, if you're sending limit-

At a glance **48**
Delving into the stack **48**
Paying your time tax **50**
Evaluating in tandem **52**
Other considerations **54**
Testing throughput **56**
For more information **58**

Go to our Web site at www.edn.com to find additional material related to this article.

ed data, maximum throughput is less important (see **sidebar** "Testing throughput"). Also, if your application isn't complex, footprint isn't critical, because you don't need a lot of headroom. Thus, my primary concern became which development environment could most efficiently help me develop my application (see **sidebar** "Other considerations").

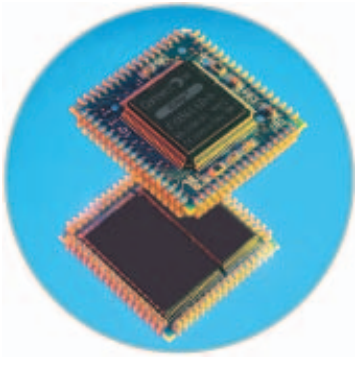
EQUIPMENT CHECKLIST

As a strong believer in a positive out-of-the-box experience, when I finally made time to look at an evaluation board, I wanted to begin the evaluation—not discover that I was expected to supply a critical piece of equipment that I didn't have handy. For example, some kits come with an Ethernet cable; some don't. If you're going to directly connect the evaluation board to your computer, however, you need a crossover cable. If you connect through a hub or router, you need a standard cable. Using the wrong cable can set you back for a while until you figure out what you've done.

One way to improve your out-of-the-box experience is to check the kit when it arrives; don't wait until the day you want to use it. One advantage of evaluating several kits is that one kit usually includes the "missing" part from another. (Before you start mixing kits, however, tag each component with a sticker, so that you remember which cable came from which kit.) One difficult-to-locate piece of equipment was an external modem that could connect to the evaluation board over a serial port, so I could test a PPP (Point to Point Protocol) connection through the phone line to an ISP (Internet-service provider). I found one at a local store for the surprising cost of $100. If you have more time to secure a modem, you can probably find one in your garage or over the Internet for less.

AT A GLANCE

- The embedded-TCP/IP market has indeed reached a point at which TCP/IP is an off-the-shelf product.
- Your evaluation is based on what you see—not what's actually there.
- One key differentiation among current systems is the available application library.
- If you work with only one evaluation kit, you'll learn whether you like it, but you won't have any idea whether you should expect more.



Connect One's iChip Internet-controller TCP/IP chip mediates the connection between a host device's processor and the Internet.

Several companies offer a variety of boards with names such as "evaluation" and "developer" or "standard" and "advanced." In some cases, the difference lies in software alone. The evaluation kit, for example, might provide only binaries and a subset of the development tools. Some differences lie in hardware: RAM-based boards, unlike flash-based boards, lose their code image when you crash and have to power down. The developer's kit may also include an emulator for setting breakpoints and capturing code traces.

In some cases, the difference in price is small between the standard and the advanced kits. In general, developing code for a standard board might take somewhat longer than it would for an advanced kit, given limited tools and lack of debugging hooks. It might be worth the extra bucks to be able to see the entire tool suite, because the tools' usefulness will almost surely be a major factor in your final selection. Some features, such as PPP, SNMP (Simple Network Management Protocol), or a real-time operating system, may be optional and require a separate purchase. You'll also often get better tech support with an advanced kit.

Most evaluation kits cost less than $500. Some kits, however, come in at several thousand dollars. In many cases, the vendor raises the price to filter out

DELVING INTO THE STACK

In previous years, network-enabling a device required intimate knowledge of the TCP/IP (Transmission Control Protocol/Internet Protocol) stack and direct system calls to manipulate and manage it. Today, TCP/IP is pretty much an off-the-shelf technology, meaning that you can link to the library, make a few calls, and have connectivity. Unless you stumble across some weird corner case of TCP/IP or stress the stack in an unanticipated way, you probably won't have to actually know anything about TCP/IP to get results. Additionally, if the vendor has provided adequate sample/demo applications, you should be able to strip down these applications to their basic components to determine the stack calls you have to make to send or receive packets, FTP (File Transfer Protocol) files, POST and GET data, and so on without understanding what any of those calls really do.

Are there reasons to delve into the stack? I've heard the most popular use of stack source code is to learn TCP/IP fundamentals. If you don't have this burning desire, then you don't necessarily need access to source code. More rare reasons include optimizing the source code to squeeze more performance from the chip either to increase the transmission rate or to free up headroom for application code; adding protocols that the stack doesn't support; implementing proprietary data schemes;, and integrating third-party code, such as security and encryption functions. In some cases, you can use a stub function—an empty function called by the stack, for example, before it sends or packetizes data—to implement such functions without understanding or breaking into the stack. An additional benefit of not breaking into the stack is that you need not recertify the stack code.

garage hobbyists, so that they can't overwhelm tech support. If you're a serious customer and have enough leverage, there's a strong chance that the vendor will dramatically drop the price. After all, too many players with solid products in this market are willing to almost give their kits away to get your business. Software vendors, on the other hand, have less flexibility with evaluations, because it's not difficult to steal code and never pay royalties for it once you've got a copy. Then again, if they won't let you use the code and tools, you can't properly evaluate the overall development environment.

GETTING STARTED

The key to a successful and efficient evaluation is proper documentation. Your evaluation is based on what you see, not what's actually there. More than one technical manual was actually an alphabetical catalog of function calls with no clear description of their relationship to each other. After a bit of searching, I usually found a readme.doc or "Getting Started" manual that provided a description of how to bring the board up. A Getting Started manual is especially important for evaluation boards running third-party software and should be written specifically for the board or chip you're evaluating.

The Getting Started manuals don't teach much. They are enough only to get you going on the board within an hour, showing you how to start the development environment and build and download a demo system. My next step was to explore on my own; I began by printing the demo source files in an attempt to figure out what the demo actually did. From this foundation, I was able to strip down the demo code to bare bones, creating a template for developing my own application and understanding how to perform basic connectivity tasks.

For me, the most important part of documentation was the sample applications. Again, I didn't want to have to learn the entire API for the TCP/IP stack just to serve up a test page (see **sidebar** "Delving into the stack"). I learn better when I'm tearing apart a working example than when I'm mulling over an explanation in a manual.

Some demos are excellent. One shows the status of DIP switches on the board. It provides a simple example without a lot of confusing bells and whistles of how to send a few bits of data dynamically to a Java applet. I ran into trouble, however, when I tried to send a different data type from the one in the demo. Because I had not yet learned the specifics of Java, I had to take a detour to the library and the Web to find a Java primer. Then, I discovered that knowing Java doesn't tell you everything; you have to use CGI (Common Gateway Interface) programs—in other words, C code—to exchange information with the Java applets. Unraveling this interconnected scheme took me longer than I wanted it to. A second and third demo that sent different types of information (or even one page that sent three data types) would have been helpful, so I could learn the differences in the functions.

Unfortunately, the range of demos that each kit includes varies, and some lack completeness. For example, in one kit, I could find neither a demo of FTP (File Transfer Protocol), which is a fairly fundamental network operation, nor a demo of sending e-mail. Kits commonly demonstrated how to send static pages but failed to show how to create or send dynamic pages. Without a demo application, I had to resort to the 300-pg manual to continue my evaluation. Thus, I found that the kits with the more

PAYING YOUR TIME TAX

It may seem like a minor issue, but your choice of interface from your PC to the evaluation board plays a significant role in determining your productivity during development. This port transfers your program and Web pages to the evaluation board. The most commonly used port is serial. Robust, dependable, and ubiquitous, the serial port is also the slowest of interfaces. As project complexity increases, downloads take longer. There's not much you can do in the two minutes it takes to download code (except perhaps read a few paragraphs of an *EDN* article). The situation starts to feel like a Vegas poker game in which the dealer shucks away a few dollars with every deal; every time you download, you lose another two minutes, and those minutes quickly add up.

Preferred interfaces are parallel or USB connections, which reduce your downloading time. At least one board I evaluated had an unpopulated space for USB. All other things being equal, if you're considering two chips, and one board has a USB port available today, and the other has a serial port ("but USB is coming"), choose the one with the USB port. Spend the hours you'll save over the course of a project with your family instead of at your workbench. For the same reason, when you develop your own hardware prototype, consider adding a higher speed port.

One problem I encountered when using a board with a parallel port stemmed from the fact that I also have a printer directly connected to my computer. Fortunately, I had a parallel-port switchbox lying around, so I didn't have to keep unplugging cables whenever I wanted to print something. For those of you using network printers, you'll run into a different but no less annoying issue: To make the evaluation board visible over Ethernet, the "Getting Started" manual advises you to put your computer on the subnet to which the evaluation board defaults. Unfortunately, doing so makes the corporate network invisible to your computer, blocking your access to the network printer as well as network services, such as e-mail. (Then again, being free from e-mail for a few hours might not be such a bad thing.) One remedy is to put a router between you and the corporate network, giving the router your IP (Internet Protocol) identity. Another is to figure out how to get the board to use DHCP (Dynamic Host Configuration Protocol) to obtain a dynamic IP address from your router.

The most efficient method I found was to change the board's fixed IP address to one that matched my computer's IP subnet. In some cases, a macro or a constant was clearly labeled in the demo program. In many demos, however, the fixed IP address was buried, and little documentation was available to help me find it. For my money, setting the fixed IP address of an evaluation board to one of my preference should be a standard part of the "getting-started" process.

complete range of sample applications were easier to learn and use and offered a more competitive edge.

From this perspective, the embedded-TCP/IP market has indeed reached a point at which TCP/IP is off-the-shelf, meaning that you can buy everything necessary to embed connectivity into a device. Given the maturity and low cost of stacks, you no longer need to develop your own stack. You can use a well-defined API to access it. However, the market is now racing to also bring the development tools and environments up to off-the-shelf standards, moving from requiring you to learn a complex API to letting you work with templates that illustrate how the API works. A proper collection of applications enabled me to create templates to learn how to use functions and to use as foundations that I could drop into my own application. Demos needn't be full-blown applications; rather, they need to illustrate key aspects of a function. To the vendors who argue that it would take only a few hours to develop the code to implement a function such as FTP, I ask: Then, why don't you have one of your engineers do it?

Thus, one key differentiation among systems is the available application library. If the demos are insufficient for building your application, you need to work the cost of learning the API and building your application from scratch (both in dollars and time to market) into your final decision. If you're still a few months out on your project, the lack of sufficient demos will probably become less of a differentiating factor as slacking vendors begin to fill out their offerings.

The RCM3000 evaluation kit from Rabbit Semiconductor comes with an evaluation board, a connectivity module, a power supply, a compiler and other application tools, a manual, board specs, and schematics.

Still out on the horizon, even for vendors with a fairly complete demo base, is further abstracting connectivity functions into a parameterized library format. Templates are great, but even when you tear them apart, you're still working at a fairly low level. Further reducing complexity to a few canned and configurable functions makes network connectivity even more accessible.

Second to the breadth of the application library is how well the code is documented and organized. Even within a single demo, the documentation often lacks consistency. Often, an editor, such as Microsoft Front Page, generates the HTML (Hyper Text Markup Language) documents, resulting in code without sufficient comments that is not formatted to read when you print it. Despite the fact that Java and C are similar, the Java class code and C code often used different commenting conventions. Some code has a full page of comments describing what it is doing, the variables it uses, and the methods into and out of the function. The other code within the same demo has comments that seem like an afterthought and offer little guidance to understanding how the code works.

Some vendors cram all the relevant demo files into a single directory. Other vendors scatter the code throughout several directories, making it more difficult to locate files or even understand which files are in use for a demo. A proper project manager lists all sources files and their dependencies with one-click access to each. In several cases, I had to break open build files to locate code modules. Connectivity applications are written in several languages, and, if the flow of code isn't documented, it's not always clear which role a function plays (and thus which language it should be written in). One feature I missed in too many instances was a simple list and description of sample code. More than once and only after complaining to vendors, I discovered that code for a function was available but buried in some nondescript directory or documented only through an obscure reference.

I appreciate it when vendors write documentation assuming that the user understands neither the API nor the details of TCP/IP. Some Getting Started manuals reveal explicitly how to set up the computer and the evaluation board, even

EVALUATING IN TANDEM

Working with several kits at one time has its drawbacks. Sometimes, I confused the function names. Also, because each vendor uses a different directory scheme for managing the various types of files–HTML (Hyper Text Markup Language), Java, C, compiled code, headers, and so on–I sometimes forgot where I should put files. This lapse led to various compilation problems, such as using the old, incorrect version of code instead of my new code.

Another difficulty was that each evaluation board used a different subnet, requiring my computer to use a different IP (Internet Protocol) address. If I wanted to try an idea between two boards, I would have to change the IP address and reboot my computer. To keep my sanity, I created a profile for each board, describing the equipment, including cable type and the setting to use on the switch box, if required, and each IP address it used. A profile serves as a checklist to ensure that you always correctly set up the board. It also works for a single evaluation board when you have to clear your desk to work on another project.

Working with multiple boards allowed me to see just how portable my code was among boards. Ideally, I should have been able to use the same Web pages and Java applets and support C code, but each board worked slightly differently with the stack. Abstracting the stack-function calls and creating my own API (application-programming interface) made it easier to port all types of code.

showing pictures. Others assume that you know what you are doing; a classic example is a manual that fails to mention that you need an Ethernet crossover cable. If you've never configured a network, even a small one, you could run into a lot of trouble. I learned only a few years ago what a subnet is—all devices on a subnet share the first three numbers of an IP address and have a distinct fourth number—when my home router disappeared from the network for two weeks and brought down the DSL line. You might consider inviting someone from IT to share your weekly doughnuts just as backup, lest you find yourself banging your head on the desk over a networking problem.

CRUNCHING CODE

Some demos offer challenging stumbling blocks. For example, to test an e-mail transmission, I needed access to a mail server. I balked at the idea of setting up a mail server on my computer. Because my deployed device would probably use an ISP, I thought I might develop my design under real-world conditions. The challenge became finding out the IP address, because you can't use the mail.isp.net format that desktops employ. You could call your ISP, although it's probably easier to send an e-mail to yourself and use a packet sniffer to track the message. Additionally, if you work in a corporate office, getting a live phone line that you can use as a modem can be challenging; the system tied to your current phone line may not support modems.

Some development tools are difficult to use because they mix Windows and command-line environments. For exam-

OTHER CONSIDERATIONS

It's easy to focus primarily on the software side of evaluation. After all, most product development is software-related, so the quality of the software tools significantly impacts your product cycle. You may find yourself shifting from the conventional view that software can compensate for hardware and be willing to force the hardware team to use its third-choice part because of the development environment. Also, given that part of the appeal of many of these chips is that they have room for your application, leave yourself some time to look beyond the connectivity libraries to evaluate the application tools as well.

Look into how the stack interacts with interrupts. A stack that uses interrupts could increase the latency of other real-time interrupts in your application. A stack that operates as a thread or separate task, however, can become resource-starved if you too infrequently yield control, resulting in lost packets.

Be aware of equipment that is optional. For example, although a few LEDs on the board could give you limited debugging feedback (for example, the green LED means you got to a certain part of the code), having an LCD screen means you can display more meaningful debugging messages to yourself. On some boards, an LCD screen is standard. On others, you can get an LCD screen as part of an optional board.

Some boards offer more than just an Ethernet interface, including 802.11 and Bluetooth interfaces. In general, the TCP/IP (Transmission Control Protocol/Internet Protocol) stack should operate over any of these physical media. Such operation depends on whether the vendor has written a driver for the protocol. Additionally, each physical layer needs drivers for the physical layer in use–even the Ethernet PHYs. If you want to use an Ethernet PHY other than the one on the evaluation board, you may have to write your own drivers or entice the vendor to do so.

Setting a device's IP address in the field can seem like a chicken-or-the-egg problem: How do you connect to the device to set its IP address if you don't have an IP address with which to connect to the device? Devices connected through a LAN can use DHCP (Dynamic Host Configuration Protocol) to obtain an IP address. Remote dial-in devices don't need an IP address per se but could call a toll-free number to obtain a local ISP (Internet-service provider) number and a log-in ID to reduce costs.

For devices connected to a network without a DHCP server, such as a home network, you can use an elegant technique sometimes called *ping-gleaning*. Because each Ethernet MAC (media-access controller) has a unique address, you can create a utility to run on a PC that asks the user for a device's "serial number"–the device's unique MAC address–and then "pings" that MAC address with an IP address. The device can then use that IP address as its own. Ping-gleaning is an example of a function that a vendor may support directly through its API (application-programming interface) but hasn't necessarily documented; if you don't know to ask about it, you might never discover its usefulness.

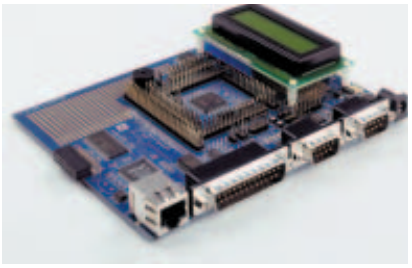
Minor differences among vendor products can make development that much easier. For example, most platforms first convert HTML files to C files. With this method, each time you add a page to your device, you must manually add the file name to the build file, insert it into your project, and add external references in your application code. With so many steps, you're bound to make a mistake every now and then and lose time trying to figure out what you missed. At least one platform allows you to add a Web page just by referencing it with a macro; the macro takes care of the rest of the work. Again, this difference isn't major, but it can tell you which vendors take ease of use seriously.

The cost of software can be a tricky subject. Vendors consider some functions optional and give them an additional price. Some pricing includes source code. Some models require upfront fees and royalties, or the software is free, meaning that the vendor has worked it into the price of the processor. Be sure to understand what your true costs will be, based on your projected volumes, across a product family and through a range of distinct product lines.

Support is an important part of both the evaluation and development cycles and can come in many forms. For example, I attended one training seminar and found that I could have covered the material on my own in half the time; however, it did give me direct access to someone who could answer my questions about embedded networks. In this light, I recommend attending a seminar or a tutorial in which you work directly with a kit at the onset of your evaluation before you invest a lot of time learning the fundamentals the hard way. Online forums dedicated to processors also exist. It's worth checking out whether a site is user- or vendor-run to determine its agenda and whether it's a hodgepodge of postings or well-organized with managed FAQs.

Security should be on your radar screen if your device will be accessible across the Internet or a public network. Though you may not require security this year, your customers may require it of you when they panic after a

ple, from the development environments, I had to run batch files in a DOS window, manage my files outside the environment, or run utilities either from a DOS window or a build batch file of my creation. Vendors could have made some of these steps more transparent—allowing the tools to launch a utility from the environment or to automatically create and manage build files. It was also difficult or annoying to complete certain simple and regular tasks. For example, one environment expects you to select the file you want to download every time, instead of defaulting to the last one you downloaded. Granted, the step was neither a major problem nor a major convenience, but my guess is that if the product developers actually used the product, they certainly would have coded such short cuts.



The eCog1 evaluation board from Cyan offers an LCD for easier debugging, an Ethernet port requiring crossover cable, code downloading over a parallel port, and a prototyping area for adding your own hardware connections.

Another aspect that takes getting used to is the use of various languages. Learning the difference between source files (index.htm) and generated files (index.c) takes some careful reading. HTML and JavaScript (which is not at all like Java) are necessary to create Web pages for the Web server (see **sidebar** "Web-page compression" on the Web version of this article at www.edn.com). Java is necessary to create classes for passing information between the Web-server and the remote monitoring browser. I also needed to run C on the embedded-networking processor to interface the hardware and the application software to the Java applets. And I needed a separate Java development environment; otherwise, I'd have to build and download my application and then serve up pages just to test simple changes (see **sidebar** "Paying your time tax"). Because these languages closely resemble each other, I had to carefully change my frame of mind when I changed languages, remembering the various subtleties and differences between them. Unfortunately, I still ended up stumped by the occasional frustrating error, because I forgot which context I was working in.

TWO KITS ARE BETTER THAN ONE

I assumed that every developer's kit had everything I was going to get; nothing else was coming to ease my job. If I had problems with simple issues, such as setup, I got nervous about the rest of the kit. However, the problem was sometimes me; I just didn't understand certain concepts the first time through. If you work with only one evaluation kit, you'll learn whether you like it, but you won't have any idea of whether you should expect more. Thus, I strongly recommend that you look at several kits. Kits cost relatively little, and the cost is effectively zero when you spread it across a high-volume device. You can donate the kits you don't use to a local university for a write-off or sell or give them to a colleague looking to expand his or her engineering skills.

Using different kits worked for me in several ways. For example, looking at dif-

high-profile embedded hack. At worst, you should understand your various options for implementing security. If the vendor doesn't have a straightforward demo or template, you may want to spend time determining just how complex it will be to later add security; to properly implement security, you'll more than likely need to break open the TCP/IP stack. One option is to create your own security scheme, which is probably sufficient if you're talking about a consumer device, such as a camera, which isn't worth hacking. But creating your own scheme has questionable value for an application such as a utility meter, whose security is worth compromising for financial reasons. One reason to use a standard, such as TCP/IP, is that any browser can query your device; using a proprietary security scheme instead of a standard, such as IPSec (Secure Internet Protocol) or SSL (Secure Sockets Layer), defeats this capability. Just remember to keep your overall goal for connectivity in mind.

Note that security does not come free. Encryption functions can have a small footprint and low performance cost, but encryption is not security. IPSec and SSL, which manage keys and sessions, require a lot of code, which makes them less suitable for 8- and 16-bit processors. If security is important to your application, you may need to consider a 32-bit processor even if you don't need high throughput.

Several times, I was tempted to figure out how to solve a problem, such as how to create dynamic Web pages, through some workaround of my own invention. For example, I thought of creating static pages with placeholders for dynamic information; I could filter for the placeholders before I transmitted information and replace them with the actual data. I don't recommend this type of experimentation, however. Doing things the proper way usually pays higher dividends; I invariably created new problems with my clever–and untested and unnecessary–workarounds.

If you're thinking of serving up a hefty user's manual from your device, consider some of these alternatives to save on memory. You could serve up a .zip file, which moves decompression from your device to the faster machine hosting the browser. It also saves on bandwidth, which is especially important across constrained connections, such as phone lines. If the computer communicating with the device has Web access, you could instead send a URL to another server. This option saves device memory and avoids many of the version-control issues inherent in "shipping" the manual in flash memory.

If you develop a useful utility, such as compressing Web pages, you might consider selling it back to the vendor. Vendors have limited resources for expanding and improving their product lines, especially in today's economy, yet every utility and sample application gives their products a competitive edge. Becoming a supplier of this nature can also give you leverage in directing the vendor's course through its product-development road map.

Most vendors have some mechanism for accepting suggestions. However, instead of focusing on how useful a new feature would be for you, explain to the vendor how the feature would benefit their entire customer base. If you can convince the vendor that your suggestion will help a broad cross section of its customer base or open new markets, you'll stand a better chance of your suggestion's making it to the top of the company's to-do list.

ferent kits showed me which aspects of designing connected devices are constant across processors and which are factors of the individual vendors. For example, once I understood how to post data to a Web page from the Web server with one kit, doing it with the next kit was a matter of figuring out which system call to make. Also, the variety of support tools and the approach-

TESTING THROUGHPUT

One goal of your evaluation may be to verify a vendor's theoretical-maximum-throughput claims. In general, the board itself shouldn't impose bottlenecks on your throughput tests; you might need to worry about this situation with higher speed connections, but if you're evaluating an 8- or a 16-bit embedded networking processor, what you see is what you get. You need to be thorough, however. For example, the vendor may have preconfigured the board to run at a reduced clock rate to conserve power.

When verifying throughput, it's important to profile the throughput based on the type of traffic you expect to see under real-world operation. Also, sending a stream of small packets will give you an idea of the aggregate overhead associated with each packet. A stream of long packets will minimize overhead, giving you a better idea of the true maximum data rate you can achieve.

Try tearing down the connection between each packet to determine how long it takes to open and close a session—an important factor for devices that will manage many other connected devices. Also, be sure to test FTP (File Transfer Protocol) function to see how the stack handles large file transfers; this figure is useful for profiling large data dumps in either direction. Sending set or pseudorandom patterns allows you to verify the accuracy of the connection.

You may also want to determine how to access performance statistics, such as how many resends the system made during your test. Finally, if you do find yourself closing in on the acceptable threshold of performance for your application, make sure that your computer or development environment, such as Windows, for example, isn't imposing the bottleneck. Run your tests from another slower or faster computer and compare results; they should be equal.

Of course, if the device serves up only 1-kbyte Web pages, connection speed doesn't matter. Or does it? If many devices are present in a network, the connection rate could matter, because connection speed determines the total time it takes to connect to a device and could affect the total number of devices you can connect to in a certain time period. The situation is analogous to watering a lawn: You might water each zone for only 15 minutes, but if you've got 10 acres of lawn, it's going to take you four days to water it all. As the number of devices you have to manage scales up, you must make a trade-off between sending less information and checking in less often.

es to performing certain types of tasks are varied. One of the most valuable benefits I discovered was that an issue that confused me when working with one kit became clear to me when working with another kit. In this light, I worked with several kits concurrently (see **sidebar** "Evaluating in tandem") and shortened my learning curve. Switching among kits also enabled me to continue my evaluation on other boards while I waited for tech support to solve a problem that had brought evaluation on one board to a standstill.

A key facet of evaluation is that it takes much less time to work on the third and fourth kits than it does on the first two. While evaluating the first two kits, I struggled with learning how to use certain aspects of TCP/IP; my learning curve with subsequent kits was significantly shorter. My questions evolved from "How do I do this?" to "How does this vendor implement this idea?" In the end, I was able to make a better final evaluation of each processor. I not only had more experience with what I could expect from a processor, but also more options to choose from.□

FOR MORE INFORMATION...

For more information on products such as those discussed in this article, go to www.edn.com/info and enter the reader-service number. When you contact any of the following manufacturers directly, please let them know you read about their products in *EDN*.

CMX Systems Inc
1-904 880-1840
www.cmx.com
Enter No. 344

Connect One
1-408-986-9602
www.connectone.com
Enter No. 345

Cyan Technology Ltd
1-781-246-4646
www.cyantechnology.com
Enter No. 346

Rabbit Semiconductor
1-530-757-8400
www.rabbitsemiconductor.com
Enter No. 347

Ubicom
1-650-210-1500
www.ubicom.com
Enter No. 348

Zilog
1-408-558-8500
www.zilog.com
Enter No. 349

SUPER INFO NUMBER
For more information on the products available from all of the vendors listed in this box, enter no. 350 at www.edn.com/info.

Reference

1. Cravotta, Nicholas, "Managing Internet-enabled devices," *EDN*, Sept 20, 2001, pg 48.

Acknowledgment

Many people at each of the companies mentioned in this article—too many to name here—offered their assistance with this article. I thank them warmly.

Author's biography

Technical Editor Nicholas Cravotta hopes to connect an entire house for his next hands-on project. You can reach him at 1-530-346-8556, fax 1-530-346-9777, or nick@edn.com.

WEB-PAGE COMPRESSION

A common characteristic of embedded applications is scarcity of memory. It may not seem obvious, but, because a Web server does not display pages, it does not need to support a JVM (Java Virtual Machine) or the 1 Mbyte of memory that a JVM requires. Depending upon how much information you want to be able to serve (you might have an extensive user's manual stored on the device), memory can get tight. Because you need to store static pages in nonvolatile memory, whether you have to worry about compression is a function of how close you come to overflowing the available memory before you have to go to the next size. However, the application itself may require enough memory that the relative size of Web pages is inconsequential; worrying about compressing Web pages doesn't yield any real gains. The same situation holds true if you need to serve up only a few pages.

Given that a nonengineer (someone not worried about memory usage) may design a product's final Web pages, it's probably best to focus on ways to process and compress finished Web pages, rather than on building compressed Web pages. For example, instead of removing white space by hand, you could build a utility that automatically removes it during the build process. Many Web designers use sloppy specialized editors that add many extraneous comments. Some comments are useful for developing pages, but you can remove them from the posted version, because source code doesn't have to be human-readable. Cropping images or reducing quality can yield easy yet significant reductions. You can also make good compression gains by replacing commonly used letter combinations with control codes and then decompressing the codes before serving up the page.

Again, compressing pages may have limited value to you if you're serving up only a few pages. Currently, compression tools aren't readily available from vendors, so you'll have to roll your own if you need to conserve memory. However, because it makes more sense for embedded devices to supply more information about and control themselves, you'll see more vendors add compression to their spec sheets.